Universidad Tecnológica de Pereira

Facultad de Ingenierías

Algoritmo propuesto para correspondencia entre imágenes

Trabajo de grado presentado por
Juan Sebastián Vega Patiño y Leiver Andrés Campeón Benjumea
como requisito parcial para optar al título de
Ingeniero de Sistemas y Computación

2017

Programa Ingeniería de Sistemas y Computación

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

Para los humanos es muy natural observar el mundo y realizar cosas como identificar objetos y formas, notar cambios de ambiente, asociar conceptos a las escenas, entender situaciones, etc; todo esto de manera aparentemente sencilla. Durante muchos años los científicos de la computación han tratado de imitar la actividad visual humana para automatizar tareas que involucran análisis de imágenes, de esta manera surgió un disciplina llamada visión por computador, la cual tiene como objetivo tomar imágenes del mundo real y realizar extracción y comprensión automática de información contenida en ellas.

Aplicaciones de visión por computador han transformado nuestro estilo de vida y han mejorado nuestra calidad de vida con por ejemplo aplicaciones en el sector de la salud [4], detección de rostros en nuestras redes sociales [15], construcción automática de modelos 3D basados en distintas fotografías de un mismo objeto [8], localización y reconocimiento de texto en fotografías en tiempo real [16], visión para la revolución automovilística que han creado los autos autónomos [3], entre muchas otras.

Hallar correspondencias entre imágenes es un procedimiento fundamental en muchas aplicaciones de visión por computador, este consiste básicamente en determinar que partes de una imagen de "referencia"se encuentran en otra de "consulta". Éste problema ha sido bien estudiado,entre los últimos aportes podemos mencionar: El método más sencillo, consiste en definir una medida de distancia entre características y comparar cada punto de la primera imagen con cada punto de la segunda, por ejemplo la propuesta de los creadores de SIFT [13], el método propuesto por

Leordeanu et al. [11] donde se optimiza una función objetivo cuadrática utilizando un algoritmo de tipo gradiente; además en los últimos años se ha intentado resolver el problema planteando una representación de las imágenes mediante grafos o hipergrafos, en cuyo caso los vértices o nodos serían las características visuales extraídas de las imágenes y las aristas la relación entre éstas, así se plantea un problema de correspondencia entre grafos. Sin embargo desde que se demostró que el problema de *graph matching* es NP hard[5], se ha intentado relajar las restricciones de correspondencia; entre lo métodos que usan grafos podemos destacar el de Zhou y De la Torre [21], el cual propone un *deformable graph matching* (DGM) cuya idea clave es una nueva factorización de la matriz de afinidad binaria del grafo y el trabajo de Duchenne et al. [7] donde hace un planteamiento de un problema de correspondencia entre hipergrafos en el cual propone solución a un problema de optimización de tensores de tercer orden.

Hasta el momento no hay solución definitiva para el problema de correspondencia y sigue siendo tema de investigación. Muchos algoritmos actualmente basan su análisis en alguna medida de distancia entre características visuales extraídas de las imágenes (hablaremos más sobre características en el capitulo 3) y a partir de allí algunos realizan un pre-procesamiento o post-procesamiento para obtener mejores resultados; entonces creemos que el punto más relevante se encuentra en la manera en que se comparan las características visuales de las imágenes, por lo tanto nosotros proponemos unas medidas de distancia basadas en propiedades geométricas (Capitulo 5) que pueden ser combinadas con medidas tradicionales para obtener correspondencias más precisas. Para ello hemos decidimos tomar figuras triangulares debido a sus propiedades y facilidad de manejo, de esta manera primero construimos una triangulación de Delaunay a partir de las características extraídas para cada una de las dos imágenes que se quieren corresponder(Capitulo 4), luego calculamos las distancias propuestas para pasar a realizar un primera búsqueda de correspondencias entre triángulos usando el algoritmo *k-nearest neighbors* (Capitulo 6), después hacemos un alineamiento entre puntos de los triángulos relacionados anteriormente, para finalmente obtener la correspondencia definitiva entre características(última sección del capitulo 6). Por último realizamos algunas pruebas con imágenes transformadas y comparamos los resultados con un algoritmo implementado en la librería para visión

por computador OpenCV[9]

Capítulo 2

Conceptos básicos

2.1. Visión por computador

La visión en el mundo animal ha evolucionado de tal manera que la capacidades adquiridas hoy por hoy son extremadamente complejas y están preparadas para maximizar la información extraída de las escenas que suceden a nuestro alrededor. La ciencia de visión por computador se ha enfocado en imitar desde hace décadas esta habilidad encontrando con que no es una tarea para nada fácil, sin embargo los avances han sido grandes y han permitido crear muchas aplicaciones que nos ayudan día a día con tareas muy variadas.

Entre todos los problemas principales de visión por computador, hallar correspondencias entre imágenes es un problema fundamental para el desarrollo de muchas otras tareas más elaboradas, aún así este no ha sido totalmente solucionado aunque los avances actuales son bastante robustos, esto se debe a que existen varias dificultades, como: trasformaciones geométricas(escalamiento, translación, etc), cambios fotométricos bruscos entre imágenes, variedad intercalase para objetos, es decir muchas formas para el mismo objeto conceptual, oclusión entre objetos en una escena. El problema se puede plantear de la siguiente manera: Teniendo I_1 y I_2, las cuales son imágenes de las se quieren hallar correspondencias, se extraen N_1 características visuales de I_1 y N_2 características de I_2 (No es necesario que $N_1 = N_2$); Se define p_i como el punto característico i-ésimo de I_1 y q_j como el punto j-ésimo de I_2

Como restricciones tenemos que una característica de I_1 solo pueden corresponder

con un punto de I_2, pero un punto de I_2 puede ser emparejado por varios puntos de I_1. Con esto decimos que el objetivo final es obtener una matriz definida así:

$$\mathcal{X} = \{X \in \{0,1\}^{N_1 \times N_2}, \forall i_1, \sum_{i_2=1}^{N_2} X_{i_1 i_2} = 1\}$$

Donde $\mathcal{X}$ es una matriz binaria en la cual $\mathcal{X}_{i,j} = 1$ si el punto p_i corresponde con el punto q_j, si no entonces $\mathcal{X}_{i,j} = 0$.

2.2. Machine learning

Machine learning (ML) es la ciencia de especificarle a las computadoras cómo aprender de un conjunto de datos a realizar una tarea especifica, sin programarla explícitamente de la manera tradicional. Una definición formal podría ser la de Tom Mitchell.

"Un programa se dice que aprende de la experiencia E respecto a alguna tarea T y alguna medida de eficacia P si su eficacia en T, medida por P, mejora con la Experiencia E".

Hoy en día muchas de nuestras herramientas pueden ser posibles gracias a ML, por ejemplo cuando subimos una foto a alguna red social y la plataforma automáticamente identifica las personas presentes en la imagen, todo eso es gracias a *machine learning*. Las imágenes que usa el algoritmo para aprender a reconocer personas es llamado conjunto de entrenamiento

Existen tres grandes categorías dentro de *Machine learning*: aprendizaje supervisado, aprendizaje no supervisado y aprendizaje por refuerzo.

- Para el supervisado se incluye el resultado esperado a los datos de entrenamiento, a esos resultados se les llama etiquetas (*labels*), por ejemplo si se quiere entrenar un modelo para detectar correos *spam*, el conjunto de datos de entrenamiento sería un montón de correos electrónicos, cada uno acompañado con una etiqueta que indica si es un correo *spam* o no.

- En el entrenamiento no supervisado sólo pasamos el conjunto de datos de entrenamiento con el objetivo de hallar patrones o relaciones entre los datos

que nos permitan identificar clases o grupos y otra clase de información útil acerca de las relaciones entre los datos.

- Aprendizaje por refuerzo toma un enfoque diferente a los dos anteriores pero similar al proceso de aprendizaje humano; consiste en un conjunto de acciones posibles, cada acción es ejecutada y juzgada para definir una recompensa o un castigo en términos del desempeño, haciendo que el modelo aprenda a seguir una serie de acciones que maximicen el desempeño final de la tarea realizada.

Capítulo 3

Detección y descripción de características

Para muchos algoritmos de visión por computador el primer paso consiste en seleccionar un conjunto de características visuales que represente la imagen que se quiere analizar, este conjunto dependerá del contenido de cada imagen y se espera que si existen imágenes similares sus conjuntos de características sean parecidos también. Ahora, las características pueden ser cualquier entidad que sea relevante para resolver el problema que se este abordando y que permita tratarlo de manera más sencilla. La definición de relevante puede variar mucho y existe una variedad de características que se usan regularmente, desde los simples píxeles, pasando por segmentos de la imagen, los colores destacados de una imagen, las formas en la imagen y *bag of words* [20], hasta otros puntos característicos más elaborados como SIFT [14], SURF [1], ORB [19] o FAST [18].

Para nuestro problema queríamos encontrar puntos de interés que cumplieran por lo menos con lo siguiente:

- Detecta el mismo punto independientemente de la imagen
- Las características son invariantes ante algunas transformaciones geométricas (escalamiento, rotación y translación) y ante algunas transformaciones fotométricas(cambios de brillo y exposición).
- Existe una manera de comparar dos características y determinar su parecido

con otra característica mediante una métrica de similitud o distancia.

- Detecta suficientes puntos para obtener una buena representación de la imagen

La familia de algoritmos basados en histogramas de la dirección de los gradientes (e.g. SIFT y SURF) cumplen con requisitos mencionados, estos métodos describen detectores de puntos característicos que son invariantes ante algunas transformaciones y también proveen estructuras que describen las características encontradas, esas estructuras son únicas para cada punto y permiten comparar dos características y definir su similitud.

En particular en este trabajo se usa SURF debido a que provee invariabilidad ante las transformaciones que se mencionaron antes, y pude ser calculado bastante rápido según sus autores [1]. De esta manera se representa cada imagen mediante el conjunto de puntos característicos detectados con SURF, junto con sus respectivos descriptores.

En la figura 3.1 se puede ver un ejemplo de puntos característicos extraídos con SURF.

Figura 3.1: Puntos característicos detectados usando SURF

Capítulo 4

Triangulación

Una triangulación consiste en la división de una superficie en un conjunto de triángulos, usualmente con la restricción de que cada lado de los triángulos es completamente compartido por dos triángulos adyacentes, evitando que los lados se superpongan y se modele la superficie mejor. Dicho lo anterior y con el objetivo en mente de introducir propiedades geométricas dentro de nuestro análisis, hemos decidido formar triángulos con los puntos característicos extraídos, usando la triangulación de Delaunay para construirlos de manera inteligente.

Se hace uso de la triangulación de Delaunay debido a que posee algunas características que la hacen invariante ante transformaciones que mantienen la posición relativa de los puntos [10](como es el caso de la rotación, translación o escalamiento), es decir, que si tenemos un conjunto de puntos al cual le aplicamos una transformación que preserve la posición relativa, entonces sus triangulaciones son iguales. Todo esto permitiría lo siguiente cuando se aplica el algoritmo de extracción y detección de características SURF a un par de imágenes similares, obtendrá dos conjuntos de puntos característicos con los mismos elementos, digamos $V^1 = V^2$, por lo tanto si calculamos las triangulaciones de la imagen original y la transformada, es decir $T(V^1)$ y $T(V^2)$, obtendríamos las mismas triangulaciones ($T(V^1) = T(V^2)$) también; por lo tanto, al calcular la similitud usando alguna métrica como la propuesta en este trabajo, se espera que un par de triángulos con la máxima similitud contengan los mismos puntos característicos como vértices y se pueda hacer una correspondencia legítima.

Un ejemplo de la triangulación de Delaunay puede observarse en la Figura 4.1

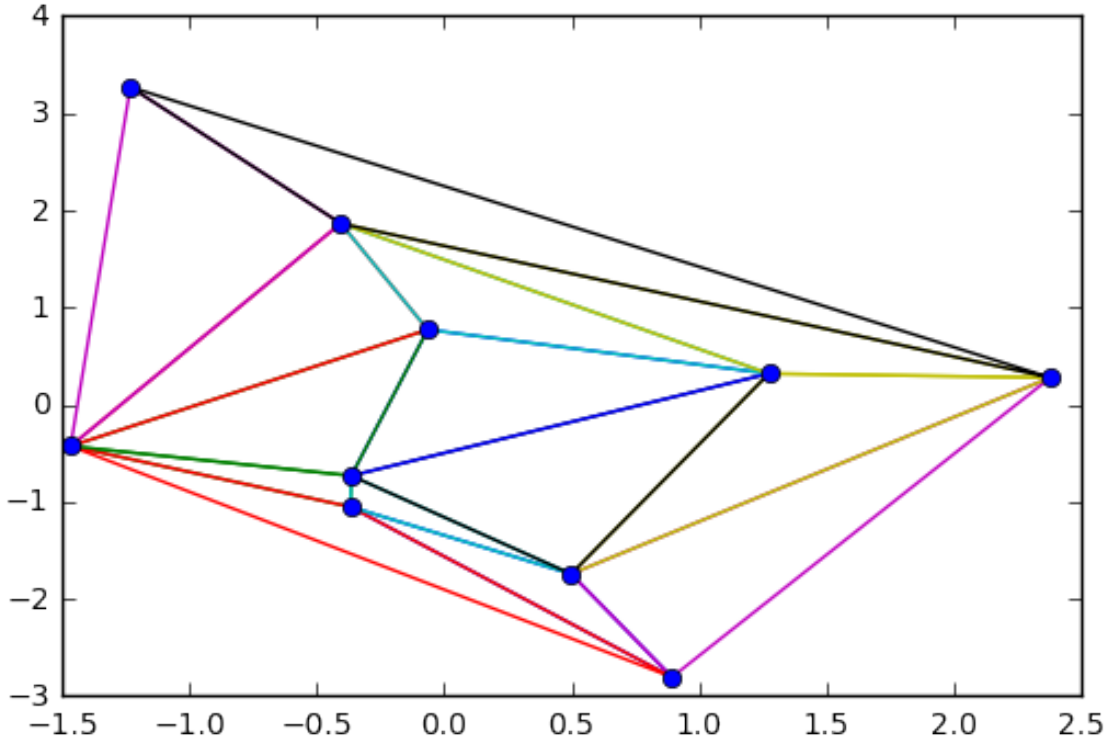

Figura 4.1: Triangulación de Delaunay para 10 puntos aleatorios.

Capítulo 5

Medidas de distancia

En el capítulo anterior vimos cómo dividir el espacio de una imagen por medio de una triangulación de sus puntos característicos. Si las imágenes corresponden a la misma escena pero una de ellas está rotada, trasladada o escalada, esperaríamos que sus respectivas triangulaciones sufrieran la misma transformación. Para aprovechar esto definimos unas medidas de distancia entre triángulos para más adelante resolver el problema de correspondencia encontrando las parejas de triángulos "más cercanos" entre las imágenes.

5.1. Distancia geométrica

Los triángulos tienen ciertas propiedades interesantes que son útiles para compararlos. En particular, estamos interesados en sus ángulos y la razón de sus lados, ya que estas dos medidas no cambian cuando un triángulo es rotado, escalado o trasladado.

Dos triángulos son similares si y sólo si sus ángulos son iguales, así que podemos calcular algún tipo de distancia entre sus ángulos y usarla en una medida de similitud. En particular tomaremos esta distancia entre dos triángulos T_i y T_j como

$$\Delta_\alpha(T_i, T_j) = \frac{1}{3}\sum_{k=1}^{3} |\sin\alpha_{ik} - \sin\alpha_{jk}| \tag{5.1}$$

Nótese que la definición de Δ_α es un promedio, esto porque la media aritmética

castiga valores muy grandes, y entre más se parezcan dos ángulos, más pequeña será la diferencia entre sus senos.

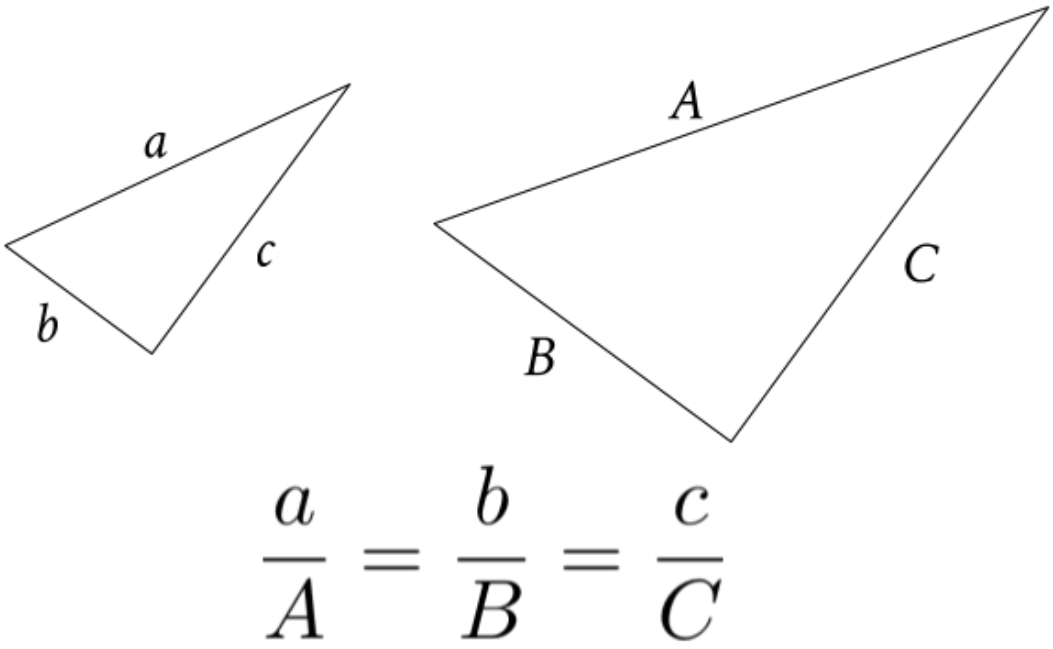

Figura 5.1: Visualización de triángulos similares por la razón de sus lados

Al mismo tiempo, dos triángulos son similares si y sólo si las razones de sus lados son iguales, como puede verse en la figura 5.1. La distancia asociada con estas razones es

$$\bar{R} = \frac{1}{3}\sum_{k=1}^{3}\frac{l_{ik}}{l_{jk}}$$
$$\Delta_R(T_i, T_j) = \sqrt{\frac{1}{3}\sum_{k=1}^{3}\left(\frac{l_{ik}}{l_{jk}} - \bar{R}\right)^2} \tag{5.2}$$

donde l_{ik} es el largo del lado opuesto al k-ésimo punto de T_i.

Esta vez, Δ_R es una desviación estándar porque ésta calcula qué tan cercanos están los valores entre sí. Como para triángulos similares las razones entre sus lados deben ser iguales, la desviación estándar tenderá a cero.

5.2. Distancia basada en descriptores

El algoritmo SURF no sólo provee los puntos característicos de una imagen, sino también una descripción vectorial única y robusta de estos puntos [1]. Muchos métodos para correspondencia entre imágenes usan este descriptor para asociar las características de las imágenes comparadas [14] de acuerdo a una medida de distancia entre sus descriptores, y nuestro enfoque no es la excepción. Definimos la distancia basada en descriptores como

$$\Delta_d(T_i, T_j) = \frac{1}{3}\sum_{k=1}^{3} \|d_{ik} - d_{jk}\| \tag{5.3}$$

Al igual que en la distancia entre ángulos, la distancia entre dos descriptores Δ_d es la media aritmética para castigar distancias euclidianas entre los vectores que sean muy grandes, tomando los vectores como puntos multi-dimensionales.

5.3. Distancia propuesta entre triángulos

Finalmente, podemos combinar las tres medidas de distancia definidas en una, obteniendo una forma de calcular distancias entre dos triángulos T_i y T_j

$$\Delta(T_i, T_j) = \sqrt{\Delta_\alpha(T_i, T_j)^2 + \Delta_R(T_i, T_j)^2 + \Delta_d(T_i, T_j)^2} \tag{5.4}$$

Un detalle de implementación que vale la pena recalcar es el alineamiento de los puntos de los triángulos a comparar. En 5.1, 5.2 y 5.3 vemos sumas sobre un índice k. Existen seis formas de alinear los triángulos, ya que cada uno está compuesto de tres puntos y en el cálculo de cada distancia se asocian de a dos puntos para cada término de la suma. La pregunta es ¿Cuál es la alineación correcta? En nuestra implementación probamos las seis alineaciones y simplemente elegimos aquella con la que se obtiene la menor distancia.

Capítulo 6

Correspondencia entre puntos característicos

Teniendo una forma de medir la distancia entre los triángulos, podemos computar una correspondencia entre las dos imágenes encontrando el triángulo más cercano en la otra imagen para cada triángulo en una imagen (en la métrica propuesta). Uno podría pensar en el algoritmo 1.

Algoritmo 1: Algoritmo simple para encontrar la correspondencia entre triangulaciones de dos imágenes

Input: R – Reference image
Input: Q – Query image
Output: M – Triangle match between the images

```
1  foreach T_i ∈ R do
2      d = ∞
3      M_i = −1
4      foreach T_j ∈ Q do
5          d_j = Δ(T_i, T_j)
6          if d_j < d then
7              d = d_j
8              M_i = j
9          end
10     end
11     M = M ∪ (i, M_i)
12 end
```

En palabras, recorremos todos los triángulos de una imagen (llamada R) y calculamos para cada uno la distancia con los triángulos de la segunda imagen (llamada Q). Elegimos como un *match* para el triángulo en R al triangulo en Q que minimice la medida de distancia definida en 5.4.

El principal problema con esta forma de calcular la correspondencia es su complejidad $\mathcal{O}(nm)$, donde n es el número de triángulos en R y m el número de triángulos en Q. Dada la gran cantidad de puntos característicos que puede detectar SURF, cada prueba con un par de fotografías podría tardar varios minutos, hasta horas con este método.

Una forma más eficiente de encontrar el *match* entre las imágenes es utilizar una familia de algoritmos que resuelven el problema de *nearest neighbour search* (NNS). Éste es un problema de optimización que consiste en encontrar el punto perteneciente a un conjunto que es más cercano a otro punto dado. La cercanía típicamente se mide expresa en términos de una medida de distancia arbitraria (como es nuestro caso), aunque usualmente se usa la distancia euclidiana. Los métodos basados en *nearest neighbour search* se conocen como algoritmos de *machine learning* no-generalizables, puesto que simplemente "recuerdan" todos los datos de entrenamiento que se le pasan (posiblemente transformados en alguna estructura de datos que facilite la indexación como un *k-d tree* o un *ball tree*).

Para nuestra implementación hicimos uso de una librería de *machine learning* de Python llamada scikit-learn [17], que puede hacer uso de las estructuras de datos mencionadas anteriormente. Luego de la construcción de la estructura de indexación, cada búsqueda del vecino más cercano a un punto tiene una complejidad de $\mathcal{O}(\log n)$, donde n es el número de puntos con el que se contruyó la estructura.

6.1. *k-d tree*

Un *k-d tree* es un árbol binario donde cada nodo es un punto de k dimensiones. Podemos pensar que cada nodo que no sea una hoja del árbol genera un hiperplano que divide el espacio en dos partes. Los puntos a la izquierda de este hiperplano se representan con el subárbol izquierdo de ese nodo, y los puntos a la derecha del hiperplano se representan por el subárbol derecho. La dirección del hiperplano se elige

de la siguiente manera: cada nodo del árbol se asocia con una de las k dimensiones, con el hiperplano perpendicular al eje de dicha dimensión. Así que, por ejemplo, si para un corte en particular se elige el eje x, todos los puntos en el subárbol con un valor más pequeño en x que el nodo aparecerán en el subárbol izquierdo, y todos los puntos con un valor más grande en x estarán en el subárbol derecho. En ese caso, el hiperplano sería definido por el valor en x del punto del nodo, y su vector normal sería el vector unitario $\hat{x}$ [2].

6.2. *Ball tree*

Un *ball tree* es un árbol binario donde cada nodo define una hiperesfera de dimensión D, conteniendo un subconjunto de los puntos a ser buscados. Cada nodo interno del árbol particiona los puntos en dos conjuntos disyuntos que son asociados con diferentes hiperesferas. Mientras que las hiperesferas en sí se pueden interceptar, cada punto es asignado a una u otra hiperesfera en la partición de acuerdo a su distancia al centro de la hiperesfera. Cada nodo hoja del árbol define una hiperesfera y enumera todos los puntos dentro de ésta.

Cada nodo en el árbol define la hiperesfera más pequeña que contiene todos los puntos en su subárbol. Esto da lugar a la propiedad de que, dado un punto de prueba t, la distancia a cualquier punto en una hiperesfera B del árbol es mayor o igual a la distancia de t a la hiperesfera [12].

Usando técnicas de *nearest neighbour search* podemos realizar la correspondencia entre las triangulaciones de las imágenes en $\mathcal{O}(m \log n)$, una mejora substancial en complejidad temporal. El método sería algo como el algoritmo 2.

6.3. Alineamiento de puntos

Una vez teniendo las parejas de triángulos que hicieron *match* debemos realizar una correspondencia entre los puntos que conforman los triángulos. En el capítulo sobre las medidas de distancia mencionamos que al calcular la distancia entre dos triángulos se alinean sus puntos minimizando la distancia propuesta. Esta misma

Algoritmo 2: Algoritmo usando NNS para encontrar la correspondencia entre triangulaciones de dos imágenes

Input: R – Reference image
Input: Q – Query image
Output: M – Triangle match between the images
1 $N = \text{NearestNeighbors}(\text{data} = R, \text{distance} = \Delta)$
2 **foreach** $T_j \in Q$ **do**
3 $\quad i = N.\text{neighbor}(T_j)$
4 $\quad M = M \cup (i, j)$
5 **end**

alineación se usa para realizar el *match* entre los puntos del triángulo. Por ejemplo, si tenemos dos triángulos $T_1 = p_1, p_2, p_3$ y $T_2 = q_1, q_2, q_3$ que forman un *match* y el algoritmo de cálculo de distancia alineó los puntos así: $(p_1, q_2), (p_2, q_1), (p_3, q_3)$; entonces el *match* de puntos sería el mismo alineamiento pero indexado según las imágenes y no según cada triángulo, es decir (p_i, q_j) donde $0 \leq i < n$, $0 \leq j < m$, y n y m son el número de puntos característicos encontrados en las imágenes. También se debe tener en cuenta que una pareja de puntos (p, q) sólo debe aparecer una vez en el conjunto de *matchs*.

Capítulo 7

Pruebas y resultados

Para probar la eficacia del algoritmo propuesto usamos un conjunto de fotografías tomadas a diferentes objetos. El conjunto de datos se compone de una fotografía "original", una fotografía del mismo objeto rotada al cambiar el ángulo de elevación de la base del trípode de la cámara (una Nikon D3100 fue usada), otra fotografía del objeto escalada al cambiar la distancia focal de la cámara, y otra combinando ambos efectos. Abajo se muestran algunos ejemplos de estas pruebas en las figuras 7.1, 7.2 y 7.3, donde las correspondencias se representan por una línea que conecta los puntos característicos respectivos.

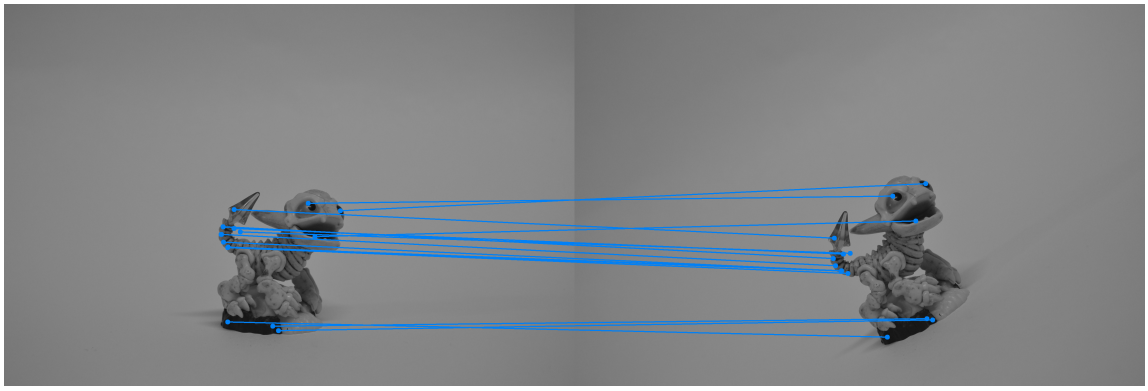

Figura 7.1: Correspondencia al comparar la imagen original con una rotada, usando pocos puntos característicos

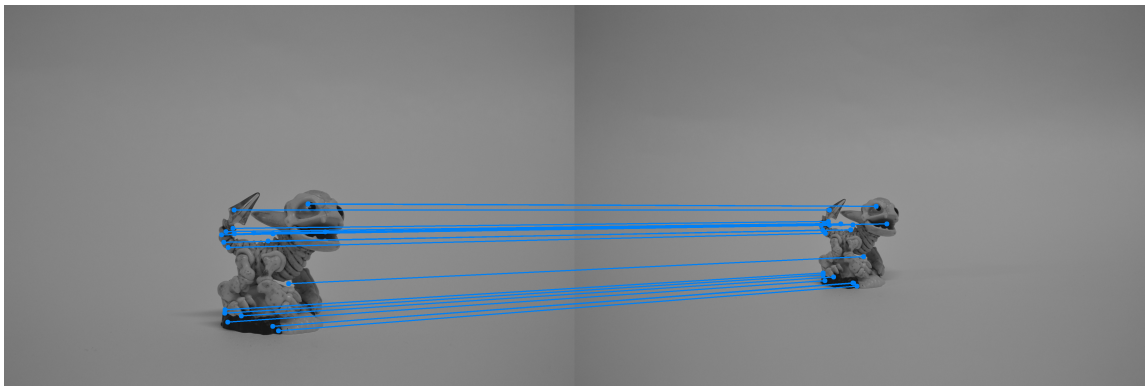

Figura 7.2: Correspondencia al comparar la imagen original con una escalada, usando pocos puntos característicos

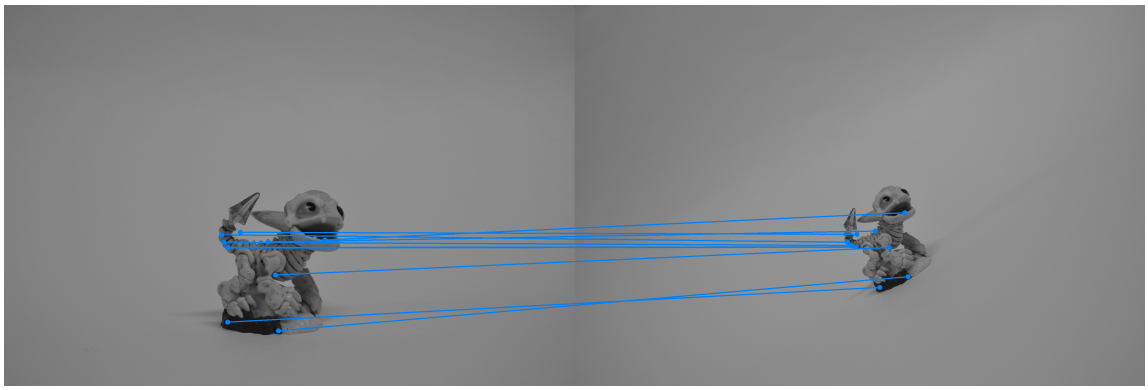

Figura 7.3: Correspondencia al comparar la imagen original con una escalada y rotada, usando pocos puntos característicos

Para cada imagen de referencia y una versión rotada y escalada de ella calculamos los puntos característicos y los descriptores usando SURF, SIFT y ORB, tomando máximo 500 puntos característicos. Luego usamos el algoritmo propuesto basado en triángulos (TBMatcher en la tabla 7.1) para SURF y un *matcher* basado en Fast Library for Approximate Nearest Neighbors (FLANN) para SURF de nuevo y el resto de detectores, como está implementado en la librería de *computer vision* OpenCV [9]. Medimos el desempeño de cada algoritmo con una prueba sencilla de *inliers/outliers* en el conjunto de imágenes descrito arriba. La prueba es la misma que la que fue realizada por el equipo que desarrolló ORB [19] y se realiza de la siguiente manera:

1. Se elije un punto de referencia V_0.
2. Para cada V_i, se encuentra una homografía H_{i0} que mapea $V_i \to V_0$.
3. Se usa H_{i0} como la transformación real a la que fue sometida la imagen y se mide el porcentaje de *inliers* para cada algoritmo.

Los resultados se muestran en la tabla 7.1.

	inlier %	#puntos (R, Q)
Monstruo		
TBMatcher	92.105	(480, 414)
SURF	78.000	(500, 424)
SIFT	87.847	(500, 444)
ORB	81.481	(500, 500)
Quijote		
TBMatcher	77.778	(485, 483)
SURF	60.465	(500, 500)
SIFT	41.667	(500, 500)
ORB	73.737	(500, 500)

Cuadro 7.1: Resultados de las pruebas sobre fotografías de una figura de un monstruo y una del Quijote

El método propuesto supera en desempeño a SURF, SIFT y ORB en las imágenes tomadas. Notando que los descriptores SIFT tienden a ser mejores en los casos de imágenes con transformaciones leves.

Capítulo 8

Conclusiones y trabajos futuros

La triangulación de los puntos característicos provee una gran simplificación del problema de correspondencia. Al abstraer este problema en lugar de trabajar directamente con la representación matricial de la imagen, pudimos obtener fácilmente una medida de distancia. Esto combinado con la elección de la triangulación de Delaunay para la partición del espacio de los puntos característicos nos permitió tomar ventaja de las propiedades geométricas de la imagen y sus invariantes ante las transformaciones consideradas.

Habiendo visto lo bien que esta abstracción funcionó al implementar y probar nuestro algoritmo en las imágenes de muestra, confirmamos que la triangulación de Delaunay es un paso importante y prometedor en muchas aplicaciones de visión por computador, especialmente aquellas que se benefician de los invariantes geométricos de los triángulos.

Propusimos una nueva forma de medir la similitud para triángulos individuales en las imágenes, combinando tanto las propiedades geométricas de los triángulos formados por las características visuales de la imagen y sus descriptores SURF.

Un problema con el algoritmo propuesto es que construir un *matcher* basado en triangulaciones no toma en cuenta proyecciones: una transformación tridimensional más general, mientras que las transformaciones consideradas ocurren en dos dimensiones. Una posible mejora a este método es definir similitudes geométricas que sean invariantes incluso cuando la imagen objetivo sea una proyección de la imagen de referencia (e.g. una fotografía de un mismo objeto tomada desde sitios diferentes).

En el capítulo sobre *Nearest Neighbours* vimos que el algoritmo "simple" de complejidad $\mathcal{O}(nm)$ no era muy eficiente en tiempo para el problema dado. Sin embargo este algoritmo es altamente paralelizable, pues cada cálculo de distancia entre triángulos es independiente entre sí. Se puede considerar encontrar el triángulo en la imagen objetivo que minimiza la distancia para cada triángulo en la imagen de referencia en un proceso separado y unir los resultados siguiendo una estrategia similar a MapReduce [6]. Así que otra posible mejora a explorar sería la paralelización del algoritmo $\mathcal{O}(nm)$ usando algún sistema de alto desempeño como un *cluster*, *grid* o GPUs.

Bibliografía

[1] Herbert Bay, Tinne Tuytelaars, y Luc Van Gool. Surf: Speeded up robust features. *Computer vision–ECCV 2006*, págs. 404–417, 2006.

[2] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–517, 1975.

[3] Mariusz Bojarski, Davide Del Testa, Daniel Dworakowski, Bernhard Firner, Beat Flepp, Prasoon Goyal, Lawrence D. Jackel, Mathew Monfort, Urs Muller, Jiakai Zhang, Xin Zhang, Jake Zhao, y Karol Zieba. End to end learning for self-driving cars. *CoRR*, abs/1604.07316, 2016. URL http://arxiv.org/abs/1604.07316.

[4] Dan C Cireşan, Alessandro Giusti, Luca M Gambardella, y Jürgen Schmidhuber. Mitosis detection in breast cancer histology images with deep neural networks. En *International Conference on Medical Image Computing and Computer-assisted Intervention*, págs. 411–418. Springer, 2013.

[5] Donatello Conte, Pasquale Foggia, Carlo Sansone, y Mario Vento. Thirty years of graph matching in pattern recognition. *International journal of pattern recognition and artificial intelligence*, 18(03):265–298, 2004.

[6] Jeffrey Dean y Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[7] Olivier Duchenne, Francis Bach, In-So Kweon, y Jean Ponce. A tensor-based algorithm for high-order graph matching. *IEEE transactions on pattern analysis and machine intelligence*, 33(12):2383–2395, 2011.

[8] Michael Goesele, Noah Snavely, Brian Curless, Hugues Hoppe, y Steven M Seitz. Multi-view stereo for community photo collections. En *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, págs. 1–8. IEEE, 2007.

[9] Itseez. Open source computer vision library. https://github.com/itseez/opencv, 2015.

[10] Der-Tsai Lee y Bruce J Schachter. Two algorithms for constructing a delaunay triangulation. *International Journal of Computer & Information Sciences*, 9(3):219–242, 1980.

[11] Marius Leordeanu, Martial Hebert, y Rahul Sukthankar. An integer projected fixed point method for graph matching and map inference. En *Advances in neural information processing systems*, págs. 1114–1122. 2009.

[12] Ting Liu, Andrew W Moore, y Alexander Gray. New algorithms for efficient high-dimensional nonparametric classification. *Journal of Machine Learning Research*, 7(Jun):1135–1158, 2006.

[13] David G Lowe. Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2):91–110, 2004.

[14] David G. Lowe. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision*, 60(2):91–110, 2004. ISSN 1573-1405. doi: 10.1023/B:VISI.0000029664.99615.94. URL http://dx.doi.org/10.1023/B:VISI.0000029664.99615.94.

[15] Markus Mathias, Rodrigo Benenson, Marco Pedersoli, y Luc Van Gool. Face detection without bells and whistles. En *European Conference on Computer Vision*, págs. 720–735. Springer, 2014.

[16] Lukáš Neumann y Jiří Matas. Real-time scene text localization and recognition. En *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, págs. 3538–3545. IEEE, 2012.

[17] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos,

D. Cournapeau, M. Brucher, M. Perrot, y E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[18] Edward Rosten y Tom Drummond. Machine learning for high-speed corner detection. *Computer Vision–ECCV 2006*, págs. 430–443, 2006.

[19] Ethan Rublee, Vincent Rabaud, Kurt Konolige, y Gary Bradski. Orb: An efficient alternative to sift or surf. En *Computer Vision (ICCV), 2011 IEEE international conference on*, págs. 2564–2571. IEEE, 2011.

[20] Yin Zhang, Rong Jin, y Zhi-Hua Zhou. Understanding bag-of-words model: a statistical framework. *International Journal of Machine Learning and Cybernetics*, 1(1-4):43–52, 2010.

[21] Feng Zhou y Fernando De la Torre. Deformable graph matching. En *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, págs. 2922–2929. 2013.